

A Combinatorial, Strongly Polynomial-Time Algorithm for Minimizing Submodular Functions

Satoru IWATA ^{*} Lisa FLEISCHER [†] Satoru FUJISHIGE [‡]

July 1999; revised October 1999

Abstract

This paper presents the first combinatorial polynomial-time algorithm for minimizing submodular set functions, answering an open question posed in 1981 by Grötschel, Lovász, and Schrijver. The algorithm employs a scaling scheme that uses a flow in the complete directed graph on the underlying set with each arc capacity equal to the scaled parameter. The resulting algorithm runs in time bounded by a polynomial in the size of the underlying set and the largest length of the function value. The paper also presents a strongly polynomial-time version that runs in time bounded by a polynomial in the size of the underlying set independent of the function value.

Key words: submodular function, combinatorial optimization, strongly polynomial-time algorithm

^{*}Division of Systems Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka 560-8531, Japan. E-mail: iwata@sys.es.osaka-u.ac.jp. A part of this work is done while on leave at the Fields Institute, Toronto, Canada. Partly supported by Grants-in-Aid for Scientific Research from Ministry of Education, Science, Sports, and Culture of Japan.

[†]Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027, USA. E-mail: lisa@ieor.columbia.edu. This work done while on leave at Center for Operations Research and Econometrics, Université Catholique de Louvain, Belgium, and at the Fields Institute, Toronto, Canada. Partially supported by NSF grants INT-9902663 and EIA-9973858.

[‡]Division of Systems Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka 560-8531, Japan. E-mail: fujishig@sys.es.osaka-u.ac.jp. Partly supported by Grants-in-Aid for Scientific Research from Ministry of Education, Science, Sports, and Culture of Japan.

1. Introduction

Grötschel, Lovász, and Schrijver [14] revealed the polynomial-time equivalence between the optimization and separation problems in combinatorial optimization via the ellipsoid method. Since then, many combinatorial problems have been shown to be polynomial-time solvable by means of their framework. The problem of minimizing submodular (set) functions is among these problems. Since the ellipsoid method is far from being efficient in practice and is not combinatorial, efficient combinatorial algorithms for submodular function minimization have been desired for a long time.

A function f on all the subsets of a finite set V is called *submodular* if it satisfies

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y) \quad \forall X, Y \subseteq V.$$

We suppose that $f(\emptyset) = 0$ without loss of generality throughout this paper.

Submodular functions arise in various branches of mathematical engineering such as combinatorial optimization and information theory. There are also close connections between submodularity and convexity [12, 21]. Examples include the matroid rank function, the cut capacity function, and the entropy function. In each of these and other applications, the *base polyhedron* defined by

$$B(f) = \{x \mid x \in \mathbf{R}^V, x(V) = f(V), \forall X \subseteq V : x(X) \leq f(X)\} \quad (1.1)$$

often plays an important role, where $x(X) = \sum_{v \in X} x(v)$ for any $X \subseteq V$.

Linear optimization problems over base polyhedra are efficiently solvable by the greedy algorithm of Edmonds [4]. Thus Grötschel, Lovász, and Schrijver [14] assert that the submodular function minimization, which is equivalent to the separation problem, is solvable in polynomial time by the ellipsoid method. Later, they also devise a strongly polynomial-time algorithm within their framework using the ellipsoid method [15].

A first step towards a combinatorial strongly polynomial-time algorithm was taken by Cunningham [2, 3], who devised a strongly polynomial-time algorithm for testing membership in matroid polyhedra as well as a pseudopolynomial-time algorithm for minimizing submodular functions. Recently, Narayanan [23] improved the running time bounds of these combinatorial algorithms by a rounding technique. Based on the minimum-norm base characterization of minimizers due to Fujishige [10, 11], Sohoni [26] gave another combinatorial pseudopolynomial-time algorithm for submodular function minimization.

For the problem of minimizing a symmetric submodular function over proper nonempty subsets, Queyranne [24] presented a combinatorial strongly polynomial-time algorithm, extending the undirected minimum cut algorithm of Nagamochi and Ibaraki [22].

In this paper, we present a combinatorial polynomial-time algorithm for submodular function minimization. Our algorithm uses an augmenting path approach with reference to a convex combination of extreme points of the base polyhedron. Such an approach was first introduced by Cunningham for minimizing submodular functions that arise

from the separation problem for matroid polyhedra [2]. This was adapted for general submodular function minimization by Bixby, Cunningham, and Topkis [1] and improved by Cunningham [3] to obtain a pseudopolynomial-time algorithm.

A fundamental tool in these algorithms is to move from one extreme point of the base polyhedron to an adjacent extreme point via an exchange operation that increases one coordinate and decreases another coordinate by the same quantity. This quantity is called the exchange capacity. These previous methods maintain a directed graph on the underlying set that represents the possible exchange operations. They are inefficient since the lower bound on the size of each augmentation is too small. In traditional network flow problems, it is possible to surmount this difficulty by augmenting only on paths of sufficiently large capacity [6]. However, it has proved difficult to adapt this scaling approach to work in the setting of submodular function minimization, mainly because the amount of augmentation is determined by exchange capacities multiplied by the convex combination coefficients. These coefficients can be as small as the reciprocal of the maximum absolute value of the submodular function.

To overcome this difficulty, we augment the directed graph corresponding to allowable exchanges with the complete directed graph on the underlying set, letting the capacity of this additional arc set depend directly on our scaling parameter. This technique was first introduced by Iwata [19], who used it to develop the first polynomial-time capacity-scaling algorithm for the submodular flow problem of Edmonds and Giles [5]. This algorithm was later refined by Fleischer, Iwata, and McCormick [7] into one of the fastest algorithms for submodular flow. Our work in this paper builds on ideas in this latter paper to develop a capacity-scaling, augmenting-path algorithm for submodular function minimization. The running time of the resulting algorithm is weakly polynomial, i.e., bounded by a polynomial in the size of the underlying set and the largest length of the function value. Even under the similarity assumption that the largest function value is bounded by a polynomial in the size of the underlying set, our algorithm is faster than the best previous combinatorial, pseudopolynomial-time algorithm [3].

We then modify our scaling algorithm to run in strongly polynomial time, i.e., in time bounded by a polynomial in the size of the underlying set, independently of the largest length of the function value. To make a weakly polynomial-time algorithm run in strongly polynomial time, Frank and Tardos [8] developed a generic preprocessing technique that is applicable to a fairly wide class of combinatorial optimization problems including the submodular flow problem and testing membership in matroid polyhedra. However, this framework does not apply to submodular function minimization. Instead, we devise a combinatorial algorithm that repeatedly detects an element that belongs to every minimizer or an ordered pair of elements with the property that if the first belongs to a minimizer then the second does.

There are some practical problems, in dynamic flows [17], facility location [27], and multi-terminal source coding [9, 16], where the polynomial-time solvability relies on a

submodular function minimization routine. Goemans and Ramakrishnan [13] discussed a class of submodular function minimization problems over restricted families of subsets. Their solution is combinatorial modulo an oracle for submodular function minimization on distributive lattices. Our algorithm can be used to provide combinatorial, strongly polynomial-time algorithms for these problems.

This paper is organized as follows. Section 2 provides preliminaries on submodular functions. Section 3 presents a scaling algorithm for submodular function minimization, which runs in weakly polynomial time. Section 4 is devoted to the strongly polynomial-time algorithm. Finally, we discuss extensions in Section 5.

2. Preliminaries

We denote by \mathbf{Z} and \mathbf{R} the set of integers and the set of reals, respectively. Let V be a finite nonempty set of cardinality $|V| = n$. For a vector $x \in \mathbf{R}^V$ we define a modular function $x : 2^V \rightarrow \mathbf{R}$ by $x(X) = \sum_{v \in V} x(v)$. For each $u \in V$, we denote by χ_u the unit vector in \mathbf{R}^V such that $\chi_u(v) = 1$ if $v = u$ and $= 0$ otherwise.

Given a submodular function f with $f(\emptyset) = 0$ and its associated base polyhedron $B(f)$ as defined in (1.1), we call a vector $x \in B(f)$ a *base*. An extreme point of $B(f)$ is called an *extreme base*. A fundamental step in submodular function minimization algorithms is to move from one base x to another base x' via an *exchange operation* that increases one coordinate while decreasing another coordinate by the same amount. The maximum amount of increase that ensures $x' \in B(f)$ is called the exchange capacity. More precisely, for any base $x \in B(f)$ and any distinct $u, v \in V$ the *exchange capacity* is

$$\tilde{c}(x, u, v) = \max\{\alpha \mid \alpha \in \mathbf{R}, x + \alpha(\chi_u - \chi_v) \in B(f)\}. \quad (2.1)$$

The exchange capacity can also be expressed as

$$\tilde{c}(x, u, v) = \min\{f(X) - x(X) \mid u \in X \subseteq V \setminus \{v\}\}. \quad (2.2)$$

In general, computing $\tilde{c}(x, u, v)$ is as hard as submodular function minimization, even when x is an extreme base. However, if x is an extreme base, then for special pairs of vertices u and v , the exchange capacity $\tilde{c}(x, u, v)$ can be computed with one function evaluation as follows.

Let $L = (v_1, v_2, \dots, v_n)$ be a linear ordering of V . For any $k \in \{1, 2, \dots, n\}$, we define $L(v_k) = \{v_1, v_2, \dots, v_k\}$. Given such a linear ordering, the greedy algorithm of Edmonds [4] computes

$$y(v_i) = f(L(v_i)) - f(L(v_{i-1})) \quad (i = 1, 2, \dots, n), \quad (2.3)$$

where $L(v_0) = \emptyset$. The resulting vector $y \in \mathbf{R}^V$ is an extreme base $y \in B(f)$. Conversely, any extreme base can be generated by applying the greedy algorithm to an appropriate

linear ordering. Note that a linear ordering $L = (v_1, v_2, \dots, v_n)$ generates an extreme base y if and only if $y(L(v_i)) = f(L(v_i))$ for $i = 1, 2, \dots, n$. For any base $y \in B(f)$, a set $X \subseteq V$ is called *y-tight* if $y(X) = f(X)$. A pair (u, v) is called *eligible* for y if u immediately succeeds v in some linear ordering that generates y . The following lemma enables us to compute an exchange capacity $\tilde{c}(y, u, v)$ if (u, v) is eligible for y .

Lemma 2.1: *Let L be a linear ordering of V that generates an extreme base $y \in B(f)$. Let L' be the linear ordering obtained by interchanging u and v that are consecutive in L . Then the extreme base y' generated by L' satisfies*

$$y' = y + \beta(\chi_u - \chi_v) \quad (2.4)$$

with

$$\beta = f(L(u) \setminus \{v\}) - f(L(u)) + y(v). \quad (2.5)$$

Moreover, we have $\tilde{c}(y, u, v) = \beta$.

Proof. Equations (2.4) and (2.5) follow from the greedy algorithm (see (2.3)). By the definition (2.1) of the exchange capacity, we have $\beta \leq \tilde{c}(y, u, v)$. Since $y(L(u)) = f(L(u))$, it follows from (2.2) and (2.5) that $\beta \geq \tilde{c}(y, u, v)$. Thus we obtain $\beta = \tilde{c}(y, u, v)$. ■

We will use Lemma 2.1 to transform one extreme base into another and to update the corresponding linear ordering.

For any vector $x \in \mathbf{R}^V$, we denote by x^- the vector in \mathbf{R}^V defined by $x^-(v) = \min\{0, x(v)\}$ for $v \in V$. The following fundamental lemma easily follows from a theorem of Edmonds [4] on the vector reduction of polymatroids (see [12, Corollaries 3.4 and 3.5]).

Lemma 2.2: *For a submodular function $f : 2^V \rightarrow \mathbf{R}$ we have*

$$\max\{x^-(V) \mid x \in B(f)\} = \min\{f(X) \mid X \subseteq V\}.$$

If f is integer-valued, then the maximizer x can be chosen from among integral bases. ■

We will not use the integrality property indicated in the latter half of this lemma. Lemma 2.2 shows a min-max relation of strong duality. A weak duality is described as follows: For any base $x \in B(f)$ and any $X \subseteq V$ we have $x^-(V) \leq f(X)$. We call the difference $f(X) - x^-(V)$ a *duality gap*. Note that, if f is integer-valued and the duality gap $f(X) - x^-(V)$ is less than one for some $x \in B(f)$ and $X \subseteq V$, then X minimizes f .

3. A Scaling Algorithm

In this section, we describe a combinatorial algorithm for minimizing an integer-valued submodular function $f : 2^V \rightarrow \mathbf{Z}$ with $f(\emptyset) = 0$. We assume an evaluation oracle for the function value of f . Let M denote an upper bound on $|f(X)|$ among $X \subseteq V$. Note that we can easily compute M by $O(n)$ calls for the evaluation oracle as follows. Let y be an extreme base generated by a linear ordering L . For any $X \subseteq V$, we have $y^-(V) \leq y(X) \leq f(X) \leq \sum_{v \in V} \max\{0, f(\{v\})\}$. Thus we obtain $M = \max\{|y^-(V)|, \sum_{v \in V} \max\{0, f(\{v\})\}\}$.

3.1. Algorithm Outline

As indicated earlier, our algorithm uses an augmenting path approach to submodular function minimization [1, 2, 3]. As with previous algorithms, we maintain a base $x \in B(f)$ as a convex combination of extreme bases $y_i \in B(f)$ indexed by $i \in I$, so that $x = \sum_{i \in I} \lambda_i y_i$. Roughly speaking, these previous algorithms use a directed graph with the arc set defined by the pairs of vertices that are eligible for some y_i , $i \in I$. They seek to increase $x^-(V)$ by performing exchange operations along a path of arcs from vertices s with $x(s) < 0$ to vertices t with $x(t) > 0$. The algorithms stop with an optimal x when there are no more augmenting paths. The corresponding minimizer X is determined by the set of vertices reachable from vertices s with $x(s) < 0$.

To adapt this procedure to a scaling framework, we use a complete directed graph on V with arc capacities that depend directly on our scaling parameter δ , an idea first introduced for submodular flows in [19]. Let $\varphi : V \times V \rightarrow \mathbf{R}$ be *skew-symmetric*, i.e., $\varphi(u, v) + \varphi(v, u) = 0$ for $u, v \in V$, and δ -feasible in that it satisfies capacity constraints $-\delta \leq \varphi(u, v) \leq \delta$ for every $u, v \in V$. The function φ can be regarded as a flow in the complete directed graph $G = (V, E)$ with the vertex set V and the arc set $E = V \times V$. The *boundary* $\partial\varphi : V \rightarrow \mathbf{R}$ of φ is defined by

$$\partial\varphi(v) = \sum_{u \in V} \varphi(u, v) \quad (v \in V). \quad (3.1)$$

Instead of trying to maximize $x^-(V)$ directly, we define $z = x - \partial\varphi$. Our algorithm seeks to maximize $z^-(V)$ and thereby increases $x^-(V)$.

We also maintain linear orderings L_i for $i \in I$ and extreme bases y_i generated by them. We start with an arbitrary linear ordering L on V and the extreme base $x \in B(f)$ generated by L . In addition, we start with the zero flow $\varphi = \mathbf{0}$. Thus, initially $z^-(V) = x^-(V) \geq -nM$. We seek to increase $z^-(V)$, and in doing so, obtain improvements in $x^-(V)$, via the δ -feasibility of φ .

The algorithm consists of scaling phases with a positive parameter δ . It starts with $\delta = M$, cuts δ in half at the beginning of each scaling phase, and ends with $\delta < 1/n^2$. Each δ -scaling phase maintains a δ -feasible flow φ , and uses the *residual graph* $G(\varphi) =$

$(V, E(\varphi))$ with the arc set

$$E(\varphi) = \{(u, v) \mid u, v \in V, u \neq v, \varphi(u, v) \leq 0\}. \quad (3.2)$$

Intuitively, $E(\varphi)$ consists of the arcs through which we can augment the flow φ by δ without violating the capacity constraints.

A δ -scaling phase starts by preprocessing φ to make it δ -feasible, and then repeatedly searches to send flow along augmenting paths in $G(\varphi)$ from $S := \{v \mid v \in V, z(v) \leq -\delta\}$ to $T := \{v \mid v \in V, z(v) \geq \delta\}$. Such a directed path is called a δ -augmenting path.

If there are no δ -augmenting paths, then the algorithm checks whether there is a pair (u, v) of vertices such that u is reachable from S by a path with residual capacity $\geq \delta$, v is not, and u immediately succeeds v in a linear ordering that generates y_i for some $i \in I$. We perform the appropriate exchange operation, and modify φ by creating residual capacity on (u, v) so that $z = x - \partial\varphi$ is invariant. This operation may increase the set of vertices reachable from S on paths of residual capacity $\geq \delta$. Once a δ -augmenting path is found, the algorithm augments the flow φ by δ through the path without changing x . As a consequence, $z^-(V)$ increases by δ in one iteration. This is an extension of a technique for handling exchange capacity arcs in submodular flows first developed in [7].

3.2. Algorithm Details

We now describe the scaling algorithm more precisely. Figure 1 provides a formal description.

At the beginning of the δ -scaling phase, after δ is cut in half, the current flow φ is 2δ -feasible. Then the algorithm modifies each $\varphi(u, v)$ to the nearest value within the interval $[-\delta, \delta]$ to make it δ -feasible. This may decrease $z^-(V)$ for $z = x - \partial\varphi$ by at most $\binom{n}{2}\delta$. The rest of the δ -scaling phase aims at increasing $z^-(V)$ by augmenting flow along δ -augmenting paths.

Let W denote the set of vertices reachable by directed paths from S in $G(\varphi)$. For each $i \in I$ we keep a linear ordering L_i that generates y_i . We call a vertex $v \in V \setminus W$ *active* if v is the last vertex in L_i among vertices in $V \setminus W$ that satisfies $W \setminus L_i(v) \neq \emptyset$. If v is active in L_i , we call (i, v) an *active pair*. We denote by Z the set of the current active pairs.

If $W \cap T = \emptyset$, there is no δ -augmenting path in $G(\varphi)$. Then, as long as there is an active pair (i, v) , i.e., $Z \neq \emptyset$, the algorithm repeatedly picks an active pair $(i, v) \in Z$ and applies **Push** (i, u, v) to u that succeeds v in L_i . Note that v active implies that $u \in W$.

The operation **Push** (i, u, v) starts with reducing the flow through (u, v) by $\alpha = \min\{\delta, \lambda_i \tilde{c}(y_i, u, v)\}$. The boundary $\partial\varphi$ moves to $\partial\varphi + \alpha(\chi_u - \chi_v)$. The operation **Push** (i, u, v) is called *saturating* if $\alpha = \lambda_i \tilde{c}(y_i, u, v)$. Otherwise, it is called *nonsaturating*. A nonsaturating **Push** (i, u, v) adds to I a new index k with $y_k := y_i$, $\lambda_k := \lambda_i - \alpha/\tilde{c}(y_i, u, v)$, and $L_k := L_i$. Whether the **Push** (i, u, v) is saturating or not, it updates

SFM(f):

Input: $f : 2^V \rightarrow \mathbf{Z}$

Output: $X \subseteq V$ minimizing f

Initialization:

$L_i \leftarrow$ an linear ordering on V

$x \leftarrow$ an extreme base in $B(f)$ generated by L_i

$I \leftarrow \{i\}$, $y_i \leftarrow x$, $\lambda_i \leftarrow 1$,

$\varphi \leftarrow \mathbf{0}$,

$\delta \leftarrow M$

While $\delta \geq 1/n^2$ **do**

$\delta \leftarrow \delta/2$

For $(u, v) \in E$ **do**

If $\varphi(u, v) > \delta$ **then** $\varphi(u, v) \leftarrow \delta$

If $\varphi(u, v) < -\delta$ **then** $\varphi(u, v) \leftarrow -\delta$

$S \leftarrow \{v \mid x(v) \leq \partial\varphi(v) - \delta\}$

$T \leftarrow \{v \mid x(v) \geq \partial\varphi(v) + \delta\}$

$W \leftarrow$ the set of vertices reachable from S in $G(\varphi)$

$Z \leftarrow$ the set of active pairs (i, v) of $i \in I$ and $v \in V$

While $S \neq \emptyset$, $T \neq \emptyset$ and $Z \neq \emptyset$ **do**,

While $W \cap T = \emptyset$ and $Z \neq \emptyset$ **do**,

Find an active pair $(i, v) \in Z$.

Let u be the vertex succeeding v in L_i .

Apply Push(i, u, v).

Update W and Z .

If $W \cap T \neq \emptyset$ **then**

Let P be a directed path from S to T in $G(\varphi)$.

For $(u, v) \in P$ **do** $\varphi(u, v) \leftarrow \varphi(u, v) + \delta$, $\varphi(v, u) \leftarrow \varphi(v, u) - \delta$

Update S , T , W , and Z .

Express x as $x = \sum_{i \in I} \lambda_i y_i$ by possibly smaller affinely independent subset I and positive coefficients $\lambda_i > 0$ for $i \in I$.

If $S = \emptyset$ **then** $X = \emptyset$ **else if** $T = \emptyset$ **then** $X = V$ **else** $X = W$

End.

Figure 1: A scaling algorithm for submodular function minimization.


```

Push( $i, u, v$ ):

 $\alpha \leftarrow \min\{\delta, \lambda_i \tilde{c}(y_i, u, v)\}$ 
 $\varphi(u, v) \leftarrow \varphi(u, v) - \alpha$ 
 $\varphi(v, u) \leftarrow \varphi(v, u) + \alpha$ 
If  $\alpha < \lambda_i \tilde{c}(y_i, u, v)$  then
     $k \leftarrow$  a new index
     $I \leftarrow I \cup \{k\}$ 
     $\lambda_k \leftarrow \lambda_i - \alpha / \tilde{c}(y_i, u, v)$ 
     $\lambda_i \leftarrow \alpha / \tilde{c}(y_i, u, v)$ 
     $y_k \leftarrow y_i$ 
     $L_k \leftarrow L_i$ 
 $y_i \leftarrow y_i + \tilde{c}(y_i, u, v)(\chi_u - \chi_v)$ 
    Update  $L_i$  by interchanging  $u$  and  $v$ .
 $x \leftarrow \sum_{i \in I} \lambda_i y_i$ 

```

Figure 2: Algorithmic description of the operation **Push**(i, u, v).

y_i as $y_i := y_i + \tilde{c}(y_i, u, v)(\chi_u - \chi_v)$, $\lambda_i := \alpha / \tilde{c}(y_i, u, v)$ if $\tilde{c}(y_i, u, v) > 0$, and L_i by interchanging u and v . Then the current base x moves to $x + \alpha(\chi_u - \chi_v)$. Thus $z = x - \partial\varphi$ is invariant.

Each time the algorithm applies the push operation, it updates the set W of vertices reachable from S in $G(\varphi)$ and the set Z of active pairs. If **Push**(i, u, v) is nonsaturating, it makes v reachable from S in $G(\varphi)$, and hence W is enlarged. Once v becomes reachable from S in $G(\varphi)$, it will never become active again for any $i \in I$ until the algorithm finds a δ -augmenting path or all the active pairs disappear. Note that we encounter at most n nonsaturating pushes before we find a δ -augmenting path or all the active pairs disappear. Each time the algorithm picks an active pair (i, v) and applies **Push**(i, u, v), the vertex v shifts towards the end of L_i . Hence the algorithm picks an active pair (i, v) at most n times before v enters W or $W \setminus L_i(v) = \emptyset$. At this point v becomes inactive, and remains inactive until the next augmentation. Hence, for each $i \in I$ the total time required for processing active vertices in L_i is $O(n^2)$.

We note that we could relax the definition of an active vertex to include any vertex $v \in V \setminus W$ whose immediate successor in L_i belongs to W . The correctness argument would apply without modifications. However, care is needed to obtain an efficient implementation.

If we find a δ -augmenting path, the algorithm augments δ units of flow along the path, which effectively increases $z^-(V)$ by δ . We also compute an expression for x as a convex

combination of at most n affinely independent extreme bases y_i , chosen from the current y_i 's. This computation is a standard linear programming technique of transforming feasible solutions into basic feasible solutions. If the set of extreme points are not affinely independent, there is a set of coefficients μ_i for $i \in I$ that is not identically zero and satisfies $\sum \mu_i y_i = 0$ and $\sum \mu_i = 0$. Using Gaussian elimination, we can start computing such μ_i until a dependency is detected. At this point, we eliminate the dependency by computing $\theta := \min\{\lambda_i/\mu_i \mid \mu_i > 0\}$ and update $\lambda_i := \lambda_i - \theta\mu_i$ for $i \in I$. At least one $i \in I$ satisfies $\lambda_i = 0$. Delete such i from I . We continue this procedure until we eventually obtain affine independence. Since a new index k is added to I only as a result of a nonsaturating push, $|I| \leq 2n$ after finding an augmenting path. The bottleneck in this procedure is the time spent computing the coefficients μ_i , which is $O(n^3)$ overall.

A δ -scaling phase ends when either $S = \emptyset$, $T = \emptyset$, or $Z = \emptyset$. In the last case, we have a set of vertices $W \subset V$ that are reachable from S in $G(\varphi)$ such that $W \cap T = \emptyset$.

Lemma 3.1: *If $Z = \emptyset$, then W is tight for x .*

Proof. If $Z = \emptyset$, for each $i \in I$ the first $|W|$ vertices in L_i must belong to W . Then it follows from (2.3) that $y_i(W) = f(W)$. Since $x = \sum_{i \in I} \lambda_i y_i$ and $\sum_{i \in I} \lambda_i = 1$, this implies $x(W) = \sum_{i \in I} \lambda_i y_i(W) = f(W)$. ■

3.3. Correctness and Complexity

We now investigate the number of iterations in each δ -scaling phase. To do this, we prove relaxed weak and strong dualities. The next lemma shows a relaxed weak duality.

Lemma 3.2: *For any base $x \in B(f)$ and any δ -feasible flow φ , the vector $z = x - \partial\varphi$ satisfies $z^-(V) \leq f(X) + \binom{n}{2}\delta$ for any $X \subseteq V$.*

Proof. For any $X \subseteq V$ we have $x(X) \leq f(X)$ and $\partial\varphi(X) \geq -\binom{n}{2}\delta$, and hence $z^-(V) \leq z(X) \leq f(X) + \binom{n}{2}\delta$. ■

A relaxed strong duality is given as follows.

Lemma 3.3: *At the end of each δ -scaling phase, the following (i)–(iii) hold for x and $z = x - \partial\varphi$.*

- (i) *If $S = \emptyset$, then $x^-(V) \geq f(\emptyset) - n^2\delta$ and $z^-(V) \geq f(\emptyset) - n\delta$.*
- (ii) *If $T = \emptyset$, then $x^-(V) \geq f(V) - n^2\delta$ and $z^-(V) \geq f(V) - n\delta$.*
- (iii) *If W is tight for x , then $x^-(V) \geq f(W) - n^2\delta$ and $z^-(V) \geq f(W) - n\delta$.*

Proof. When the δ -scaling phase finishes with $S = \emptyset$, we have $x(v) > \partial\varphi(v) - \delta \geq -n\delta$ for every $v \in V$, which implies $x^-(V) \geq f(\emptyset) - n^2\delta$ as well as $z^-(V) \geq f(\emptyset) - n\delta$. Similarly, when the δ -scaling phase finishes with $T = \emptyset$, we have $x(v) < \partial\varphi(v) + \delta \leq n\delta$ for every $v \in V$, which implies $x^-(V) \geq x(V) - n^2\delta = f(V) - n^2\delta$ as well as $z^-(V) \geq x(V) - n\delta$.

When the δ -scaling phase ends with $x(W) = f(W)$ due to Lemma 3.1, then $S \subseteq W \subseteq V \setminus T$ and $\partial\varphi(W) < 0$. By the definitions of S and T , we also have $x(v) > \partial\varphi(v) - \delta \geq -n\delta$ for every $v \in V \setminus W$ and $x(v) < \partial\varphi(v) + \delta \leq n\delta$ for every $v \in W$. Therefore we have $x^-(V) = x^-(W) + x^-(V \setminus W) \geq x(W) - n\delta|W| - n\delta|V \setminus W| = f(W) - n^2\delta$ as well as $z^-(V) = z^-(W) + z^-(V \setminus W) \geq x(W) - \partial\varphi(W) - |W|\delta - \delta|V \setminus W| \geq f(W) - n\delta$. ■

Lemma 3.3 implies that at the beginning of the δ -scaling phase, after δ is cut in half, $z^-(V)$ is at least $f(X) - 2n\delta$ for some $X \subseteq V$. Making the current flow δ -feasible decreases $z^-(V)$ by at most $\binom{n}{2}\delta$. Each δ -augmentation increases $z^-(V)$ by δ . Since $z^-(V)$ is at most $f(X) + \binom{n}{2}\delta$ at the end of a δ -phase by Lemma 3.2 the number of δ -augmentations per phase is at most $n^2 + n$ for all phases after the first. Since $z^-(V) = x^-(V) \geq -nM$ at the start of the algorithm, setting the initial $\delta = M$ is more than sufficient to obtain a similar bound on the number of augmentations in the first phase.

As an immediate consequence of Lemmas 2.2 and 3.3, we also obtain the following.

Theorem 3.4: *The algorithm obtains a minimizer of f at the end of the δ -scaling phase with $\delta < 1/n^2$.*

Proof. By Lemma 3.3, the output X of the algorithm satisfies $x^-(V) \geq f(X) - n^2\delta > f(X) - 1$. For any $Y \subseteq V$, the weak duality in Lemma 2.2 asserts $x^-(V) \leq f(Y)$. Thus we have $f(X) - 1 < f(Y)$, which implies by the integrality of f that X minimizes f . ■

Theorem 3.5: *Algorithm SFM runs in $O(n^5 \log(nM))$ time.*

Proof. The algorithm starts with $\delta = M$ and ends with $\delta < 1/n^2$, so the algorithm consists of $O(\log(nM))$ scaling phases. Each scaling phase finds $O(n^2)$ δ -augmenting paths. To find an augmenting path, we perform at most $O(n^2)$ pushes per extreme base. A saturating push requires $O(1)$ time while a nonsaturating one $O(n)$ time. Here, note that there are less than n nonsaturating pushes per augmenting path. Hence, the time spent in pushes per augmenting path is $O(n^3)$. After each augmentation, we also update the expression $x = \sum_{i \in I} \lambda_i y_i$, which also takes $O(n^3)$ time per augmentation. Thus the overall complexity of SFM is $O(n^5 \log(nM))$. ■

In this section, we have shown a weakly polynomial-time algorithm for minimizing integer-valued submodular functions. The integrality of a submodular function f guarantees that if we have a base $x \in B(f)$ and a subset X of V such that the duality gap $f(X) - x^-(V)$ is less than one, X is a minimizer of f . Except for this we have not used the integrality of f . It follows that for any real-valued submodular function $f : 2^V \rightarrow \mathbf{R}$,

if we are given a positive lower bound ϵ for the difference between the second minimum and the minimum value of f , the present algorithm works for the submodular function $(1/\epsilon)f$ and runs in $O(n^5 \log(nM/\epsilon))$ time, where M is an upper bound on $|f(X)|$ among $X \subseteq V$.

4. A Strongly Polynomial-Time Algorithm

This section presents a strongly polynomial-time algorithm for minimizing submodular functions using the scaling algorithm in Section 3. The new algorithm exploits the following proximity lemma.

Lemma 4.1: *At the end of the δ -scaling phase, if $x(w) < -n^2\delta$, then w belongs to every minimizer of f .*

Proof. Let X be any minimizer of f . There exists a vector $y \in B(f)$ with $x^- \leq y^-$ such that $y^-(V) = f(X)$. Note that $y(v) \geq 0$ for each $v \in V \setminus X$. By Lemma 3.3, there exists a subset $Y \subseteq V$ such that $x^-(V) \geq f(Y) - n^2\delta$. Then we have $y^-(w) - x^-(w) \leq y^-(V) - x^-(V) \leq f(X) - f(Y) + n^2\delta \leq n^2\delta$. This implies $y(w) < 0$ due to the assumption, and hence $w \in X$. ■

Let $f : 2^V \rightarrow \mathbf{R}$ be a submodular function and $x \in B(f)$ an extreme base whose components are bounded from above by $\eta > 0$. Assume that there exists a subset $Y \subseteq V$ such that $f(Y) \leq -\kappa$ for some positive parameter κ , which will be specified later as $\eta/2$. We then apply the scaling algorithm starting with $\delta = \eta$ and the extreme base $x \in B(f)$. After $\lceil \log_2(n^3\eta/\kappa) \rceil$ scaling phases, δ becomes less than κ/n^3 . Since $x(Y) \leq f(Y) \leq -\kappa$, at least one element $w \in Y$ satisfies $x(w) < -n^2\delta$. By Lemma 4.1, such an element w belongs to every minimizer of f . We denote this procedure by $\text{Fix}(f, x, \eta)$.

We now discuss how to apply this procedure to design a strongly polynomial-time algorithm for minimizing a submodular function f . If $f(V) > 0$, we replace the value $f(V)$ by zero. The set of minimizers remains the same unless the minimum value is zero, in which case we may assert that \emptyset minimizes f .

An ordered pair (u, v) of distinct vertices $u, v \in V$ is said to be *compatible* with f if $u \in X$ implies $v \in X$ for every minimizer X of f . Our algorithm keeps a directed acyclic graph $D = (V, F)$ whose arcs are compatible with f . Initially, the arc set F is empty. Each time the algorithm finds a compatible pair (u, v) with f , it adds (u, v) to F . When this gives rise to a cycle in D , the algorithm contracts the strongly connected component $U \subseteq V$ to a single vertex and modifies the submodular function f by regarding U as a singleton.

For each $v \in V$, let $R(v)$ denote the set of vertices reachable from v in D and f_v the submodular function on the subsets of $V \setminus R(v)$ defined by

$$f_v(X) = f(X \cup R(v)) - f(R(v)) \quad (X \subseteq V \setminus R(v)).$$

A linear ordering (v_1, \dots, v_n) of V is called *consistent* with D if $i < j$ implies $(v_i, v_j) \notin F$. Consider an extreme base $x \in B(f)$ generated by a linear ordering (v_1, v_2, \dots, v_n) consistent with D . The extreme base generated by a consistent linear ordering is also called *consistent*. For any consistent extreme base $x \in B(f)$, the greedy algorithm defines $x(v)$ by $x(v) = f(U) - f(U \setminus \{v\})$ for some $U \subseteq V$ with $R(v) \subseteq U$. It then follows from the submodularity of f that x satisfies $x(v) \leq f(R(v)) - f(R(v) \setminus \{v\})$ for each $v \in V$.

In each iteration, the algorithm computes

$$\eta = \max\{f(R(v)) - f(R(v) \setminus \{v\}) \mid v \in V\}. \quad (4.1)$$

If $\eta \leq 0$, then an extreme base $x \in B(f)$ consistent with D satisfies $x(v) \leq 0$ for each $v \in V$. In this case $x^-(V) = x(V) = f(V)$, which implies that V minimizes f by Lemma 2.2. If in addition $f(V) = 0$, then the original function may have had a positive value of $f(V)$. Therefore, the algorithm returns \emptyset or V as a minimizer, according to whether $f(V) = 0$ or $f(V) < 0$.

If $\eta > 0$, let u be an element that attains the maximum in the right-hand side of (4.1). Then we have $f(R(u)) = f(R(u) \setminus \{u\}) + \eta$, which implies either $f(R(u)) \geq \eta/2 > 0$ or $f(R(u) \setminus \{u\}) < -\eta/2 < 0$ holds.

In the former case ($f(R(u)) \geq \eta/2$), we have $f_u(V \setminus R(u)) = f(V) - f(R(u)) \leq -\eta/2$. The algorithm finds a consistent extreme base $x \in B(f_u)$ generated by a linear ordering (v_1, \dots, v_k) consistent with D , where $k = |V \setminus R(u)|$. That is, let $x(v_1) = f_u(\{v_1\})$ and $x(v_j) = f_u(\{v_1, v_2, \dots, v_j\}) - f_u(\{v_1, v_2, \dots, v_{j-1}\})$ for $j = 2, \dots, k$. Then the extreme base $x \in B(f_u)$ satisfies $x(v) \leq f(R(v)) - f(R(v) \setminus \{v\}) \leq \eta$. Thus we may apply the procedure $\text{Fix}(f_u, x, \eta)$ to find an element $w \in V \setminus R(u)$ that belongs to every minimizer of f_u . Since $\kappa = \eta/2$, the procedure terminates within $O(\log n)$ scaling phases. Consequently, we obtain a new pair (u, w) that is compatible with f . Hence the algorithm adds the arc (u, w) to F .

In the latter case ($f(R(u) \setminus \{u\}) < -\eta/2$), we compute an extreme base $x \in B(f)$ consistent with D by the greedy algorithm, and then apply the procedure $\text{Fix}(f, x, \eta)$ to find an element $w \in R(u)$ that belongs to every minimizer of f . Since $x(v) \leq \eta$ for every $v \in V$ and $\kappa = \eta/2$, the procedure terminates within $O(\log n)$ scaling phases. Note that every minimizer of f includes $R(w)$. Thus it suffices to minimize the submodular function f_w , which is now defined on a smaller underlying set. Figure 3 provides a formal description of the strongly polynomial-time algorithm.

Theorem 4.2: *The algorithm in Figure 3 computes the minimizer of a submodular function in $O(n^7 \log n)$ time, which is strongly polynomial.*

Proof. Each time we call the procedure Fix , the algorithm adds a new arc to D or deletes a set of vertices. This can happen at most n^2 times. Thus the overall running time of the algorithm is $O(n^7 \log n)$, which is strongly polynomial. \blacksquare

```

Input:  $f : 2^V \rightarrow \mathbf{R}$ 
Output:  $X \subseteq V$  minimizing  $f$ 

Initialization:
   $X \leftarrow \emptyset$ 
   $F \leftarrow \emptyset$ 
While  $V \neq \emptyset$  do
  If  $f(V) > 0$  then  $f(V) \leftarrow 0$ 
   $\eta \leftarrow \max\{f(R(v)) - f(R(v) \setminus \{v\}) \mid v \in V\}$ 
  If  $\eta \leq 0$  then break
  Let  $u \in V$  attain the maximum above.
  If  $f(R(u)) \geq \eta/2$  then
    Find a consistent extreme base  $x \in B(f_u)$  by the greedy algorithm.
     $w \leftarrow \text{Fix}(f_u, x, \eta)$ 
    If  $u \in R(w)$  then
      Contract  $\{v \mid v \in R(w), u \in R(v)\}$  to a single vertex.
    Else  $F \leftarrow F \cup \{(u, w)\}$ 
  Else
    Find a consistent extreme base  $x \in B(f)$  by the greedy algorithm.
     $w \leftarrow \text{Fix}(f, x, \eta)$ 
     $V \leftarrow V \setminus R(w)$ 
     $f \leftarrow f_w$ 
    Find a subset  $Q$  of the original underlying set represented by  $R(w)$ .
     $X \leftarrow X \cup Q$ 
  If  $f(V) < 0$  then
    Find a subset  $Q$  of the original underlying set represented by  $V$ .
     $X \leftarrow X \cup Q$ 

End.

```

Figure 3: A strongly polynomial-time algorithm for submodular function minimization.

5. Concluding Remarks

This paper presents a strongly polynomial-time algorithm for minimizing submodular functions defined on Boolean lattices. We now briefly discuss minimizing submodular functions defined on more general lattices.

Consider a submodular function $f : \mathcal{D} \rightarrow \mathbf{R}$ defined on a distributive lattice \mathcal{D} represented by a poset \mathcal{P} on V . Then the associated base polyhedron is unbounded in general (see [12]).

An easy way to minimize such a function f is to consider the reduction of f by a sufficiently large vector. As described in [12, p. 56], we can compute an upper bound \hat{M} on $|f(X)|$ ($X \in \mathcal{D}$). Let f' be the rank function of the reduction by a vector with each component being equal to \hat{M} . The submodular function f' is defined on 2^V and the set of minimizers of f' coincides with that of f . Thus, we may apply our algorithms. However, each evaluation of the function value of f' requires $O(n^2)$ elementary operations in addition to a single call for the evaluation of f . Consequently, this approach takes $O(n^7 \min\{\log(n\hat{M}), n^2 \log n\})$ time.

Alternatively, we can slightly extend the algorithms in Sections 3 and 4 by keeping the base $x \in B(f)$ as a convex combination of extreme bases y_i 's plus a vector in the characteristic cone of $B(f)$. The latter can be represented as a boundary of a non-negative flow in the Hasse diagram of \mathcal{P} . This extension enables us to minimize f in $O(n^5 \min\{\log(n\hat{M}), n^2 \log n\})$ time.

Submodular functions defined on modular lattices naturally arise in linear algebra. Minimization of such functions has a significant application to computing canonical forms of partitioned matrices [18, 20]. It remains an interesting open problem to develop an efficient algorithm for minimizing submodular functions on modular lattices, even for those specific functions that arise from partitioned matrices.

Independently of this work, and almost simultaneously, Schrijver has also developed a combinatorial, strongly polynomial-time algorithm for submodular function minimization [25]. His algorithm also extends Cunningham's approach. However, the resulting algorithm is quite different from ours.

Acknowledgments

We are grateful to Bill Cunningham, Michel Goemans, and Maiko Shigeno for their useful comments.

References

- [1] R. E. Bixby, W. H. Cunningham, and D. M. Topkis: Partial order of a polymatroid extreme point, *Math. Oper. Res.*, **10** (1985), 367–378.
- [2] W. H. Cunningham: Testing membership in matroid polyhedra, *J. Combinatorial Theory*, **B36** (1984), 161–188.
- [3] W. H. Cunningham: On submodular function minimization, *Combinatorica*, **5** (1985), 185–192.
- [4] J. Edmonds: Submodular functions, matroids, and certain polyhedra, *Combinatorial Structures and Their Applications*, R. Guy, H. Hanani, N. Sauer, and J. Schönheim, eds., Gordon and Breach, 69–87, 1970.
- [5] J. Edmonds and R. Giles: A min-max relation for submodular function on graphs, *Ann. Discrete Math.*, **1** (1977), 185–204.
- [6] J. Edmonds and R. Karp: Theoretical improvements in algorithmic efficiency for network flow problems, *J. ACM*, **19** (1972), 248–264.
- [7] L. Fleischer, S. Iwata, and S. T. McCormick: A faster capacity scaling algorithm for submodular flow, 1999.
- [8] A. Frank and É. Tardos: An application of simultaneous Diophantine approximation in combinatorial optimization, *Combinatorica*, **7** (1987), 49–65.
- [9] S. Fujishige: Polymatroidal dependence structure of a set of random variables, *Information and Control*, **39** (1978), 55–72.
- [10] S. Fujishige: Lexicographically optimal base of a polymatroid with respect to a weight vector, *Math. Oper. Res.*, **5** (1980), 186–196.
- [11] S. Fujishige: Submodular systems and related topics, *Math. Programming Study*, **22** (1984), 113–131.
- [12] S. Fujishige: *Submodular Functions and Optimization*, North-Holland, 1991.
- [13] M. X. Goemans and V. S. Ramakrishnan: Minimizing submodular functions over families of subsets, *Combinatorica*, **15** (1995), 499–513.
- [14] M. Grötschel, L. Lovász, and A. Schrijver: The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica*, **1** (1981), 169–197.
- [15] M. Grötschel, L. Lovász, and A. Schrijver: *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, 1988.

- [16] T.-S. Han: The capacity region of general multiple-access channel with correlated sources, *Information and Control*, **40** (1979), 37–60.
- [17] B. Hoppe and É. Tardos: The quickest transshipment problem, *Proceedings of 5th ACM/SIAM Symposium on Discrete Algorithms* (1995), 512–521.
- [18] H. Ito, S. Iwata, and K. Murota: Block-triangularization of partitioned matrices under similarity/equivalence transformations, *SIAM J. Matrix Anal. Appl.*, **15** (1994), 1226–1255.
- [19] S. Iwata: A capacity scaling algorithm for convex cost submodular flows, *Math. Programming*, **76** (1997), 299–308.
- [20] S. Iwata and K. Murota: A minimax theorem and a Dulmage-Mendelsohn type decomposition for a class of generic partitioned matrices, *SIAM J. Matrix Anal. Appl.*, **16** (1995), 719–734.
- [21] L. Lovász: Submodular functions and convexity. *Mathematical Programming — The State of the Art*, A. Bachem, M. Grötschel and B. Korte, eds., Springer-Verlag, 1983, 235–257.
- [22] H. Nagamochi and T. Ibaraki: Computing edge-connectivity in multigraphs and capacitated graphs, *SIAM J. Discrete Math.*, **5** (1992), 54–64.
- [23] H. Narayanan: A rounding technique for the polymatroid membership problem, *Linear Algebra Appl.*, **221** (1995), 41–57.
- [24] M. Queyranne: Minimizing symmetric submodular functions, *Math. Programming*, **82** (1998), 3–12.
- [25] A. Schrijver: A combinatorial algorithm minimizing submodular functions in strongly polynomial time, 1999.
- [26] M. A. Sohoni: Membership in submodular and other polyhedra. Technical Report TR-102-92, Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, India, 1992.
- [27] A. Tamir: A unifying location model on tree graphs based on submodularity properties, *Discrete Appl. Math.*, **47** (1993), 275–283.